Towards Declarative and Efficient Querying on Protein Structures

Jignesh M. Patel
University of Michigan

Biology 101

When there is something to do in the cell, it is a protein that does it.
Role of DBMS in Bioinformatics

- Large Data Sets
  - Growing exponentially!
- Data Types
  - Sequences/arrays/3-D/text
- Complex Queries
  - Ad-hoc tools today
  - Integrated querying done using procedural methods
- Scalability/Parallelism
  - Home-grown techniques often used today
- Need: Declarative and efficient querying

![Graph showing growth of DNA base pairs and sequences from 1982 to 2004.

Protein Querying

- Protein functionality determined by
  - Sequence composition
  - Geometric structure
- Need to query across all structures
- Current methods
  - Non-declarative tools, often not very efficient on large data sets
  - Support querying on only one structure
Periscope

Goal: Design, implement, and evaluate a database management system for declarative and efficient querying on all protein structures

Roadmap

► Background and Introduction
► Primary Structure Sequence Matching
► Querying on Secondary Structure
► PiQA: Integrated Query Algebra
► Summary
Sequence Matching

- Find similar sequences
  - Given a protein sequence, find homologous matches in the database
  - Similarity based on “local similarity”
  - A local-alignment algorithm
  - Operations: Replace, Delete, Insert
  - Score using a substitution matrix

**Database:** THE TRAIN DRIVER’S CABIN

**Query:** DRAIN

Score:

Target: … C    A    B    I    N …

Query: … D    R    A    I    N …

Score: …+2   -1 +3 -1  +3   +3 …
Smith-Waterman

\[ G_{ij} = \max \begin{cases} 0, \\ G_{i-1,j-1} + S(q_i \rightarrow t_j), \\ G_{i-1,j} + S(q_i \rightarrow \_), \\ G_{i,j-1} + S(\_ \rightarrow t_j) \end{cases} \]

Suffix Trees

- Compact Patricia trie
  - Every suffix has a path (to leaf)
  - Every subsequence is a prefix of a path
- Can find subsequences very fast
  - E.g. GTACG
- Size: ~10x
OASIS

- Search driven by the suffix tree
  - Fill up the S-W columns by traversing down the suffix tree
- Best-first: expand node in the tree with highest expected score
  - Expected Score = current score + best possible score for unconsumed portion of query
  - Guarantees online behavior!
- Exploits redundancy in the database

Query: TACG
Unit Edit Distance Matrix: Same Symbol Substitution = 1, else -1

Score: $2 + 1 + 1 = 4$
Score: $1 + 1 + 1 = 3$
Experimental Setup: Comparison

- Data set: swissprot
  - 110K proteins, # symbols: 40M, data size: 40MB
  - Index Size: 500MB (~12.5 bytes per symbol)
- Workload: 100 queries from ProClass motif database
  - Short queries – lengths 6 - 56, avg len = 16
- PAM30 scoring matrix
- BLAST parameters (short query settings, 5-15 residues)
  - E=20,000, word size = 2
- OASIS: minScore = \( \frac{\ln(K_{mn}) - \ln(E)}{\lambda} \)
- Platform: 1.7GHz Xeon, Linux, 256MB buffer pool, 2K page size, clock replacement policy

Experimental Results

- OASIS 1-2 orders of magnitude faster than S-W, and usually also faster than BLAST
- OASIS retrieved ~ 60% more matches than BLAST

Lots more in our VLDB’03 paper
Roadmap

► Background and Introduction
► Primary Structure Sequence Matching
► Querying on Secondary Structure
► PiQA: Integrated Query Algebra
► Summary

Querying Protein Secondary Structure

► Secondary Structure describes protein folds
  ▪ Beta sheets (e); Alpha helices (h); Loops (l)
  ▪ Predicted structure; helps determine protein function

► Data Model
  ▪ Sequence of Segments
  ▪ ‘h h h l l e e e e’ → <3h>, <2 l>, <4 e>

► Query Language
  ▪ Sequence of regular expression terms
    ▪ Example: Beta sheet of length 4-10 followed at some point by an helix of length 3-6
    Query: <e 4 10> <? 0 Inf> <h 3 6>
Schema and Query Evaluation

### Protein Table

<table>
<thead>
<tr>
<th>id</th>
<th>name</th>
<th>len</th>
<th>sec-seq</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A</td>
<td>5</td>
<td>lleee</td>
</tr>
<tr>
<td>2</td>
<td>B</td>
<td>6</td>
<td>hhhee</td>
</tr>
</tbody>
</table>

### (Derived) Segment Table

<table>
<thead>
<tr>
<th>seg-id</th>
<th>id</th>
<th>type</th>
<th>len</th>
<th>start-pos</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>l</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>e</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>h</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>e</td>
<td>2</td>
<td>4</td>
</tr>
</tbody>
</table>

### Evaluation Methods

- CSP: complex scan
- SSS: scan segment table
- ISS: use segment index
- MISS (k): multiple ISS

**Scan each protein tuple and evaluate query using a finite state automata**

**N-way merge join of k multiple segment index scans + FK join**

---

**Multiple Index Scan Method: MISS(k)**

**Query:**

\(<P_1><P_2>\ldots<P_n>\)

To satisfy other query predicates

Merge based on protein id and ordering constraints

1. Idx Scan (type, len)
2. Idx Scan (type, len)
3. \ldots
4. Idx Scan (type, len)

Sort (id, start)

Sort (id, start)

Sort (id, start)

Sort (id, start)

Idx Scan (id)

INLJ (id)

CSP

Merge

Protein Table

k most highly selective query predicates
Query Optimizer

► Chooses best method for given query
► Optimization:
  ▪ Requires estimates of query predicate selectivities and result cardinality
  ▪ Cost model: CPU and IO
► Estimation: Two types of histograms
  ▪ Basic: segment predicate selectivity
  ▪ Complex: query selectivity

Basic Histogram

► Estimates selectivity of individual query predicates
► 2-D array
  ▪ Dimensions: Length, Fold type
  ▪ Value: number of each <type, len> segment
► Example
  ▪ Pred <e 4 4>, Est: 52
  ▪ Pred <e 2 4>, Est: 35+45+52

<table>
<thead>
<tr>
<th>Len</th>
<th>H</th>
<th>E</th>
<th>L</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>20</td>
<td>10</td>
<td>18</td>
</tr>
<tr>
<td>2</td>
<td>23</td>
<td>35</td>
<td>25</td>
</tr>
<tr>
<td>3</td>
<td>36</td>
<td>45</td>
<td>33</td>
</tr>
<tr>
<td>4</td>
<td>44</td>
<td>52</td>
<td>35</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>50</td>
<td>4</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>-</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Complex Histogram

- Estimates **result** cardinality
- Four-dimensional structure
  - Protein id (equi-width buckets)
  - Start position (equi-width buckets)
  - Length (1 – 50)
  - Type (‘e’, ‘h’, ‘l’) — Same as in the basic histogram
- Example: Position [x] [y] [z] [‘e’]
  - holds the number of <e z z> segments whose starting position is in the range of the y\textsuperscript{th} bucket and whose protein id lies within the x\textsuperscript{th} bucket range

Query: {<P1> <P2>}
- Compute selectivity of query result

∑ \int \forall = \sum_{\text{start positions}} \sum_{\text{pos}=0}^{\text{start bucketNum}} \sum_{\text{bucketNum}=1}^{\text{last start bucketNum}} \left( n_{\text{P1}}^{\text{start}} \times n_{\text{P2}}^{\text{pos+1}} \right) / \text{IDbucketWidth}

∑ \sum_{\text{start positions}} \left( n_{\text{P1}}^{\text{start}} \times n_{\text{P2}}^{\text{pos+1}} \right) / \text{IDbucketWidth}
Experimental Evaluation

- Techniques implemented in Periscope
  - Configuration: Using the SHORE storage manager, 64MB buffer pool size, 16K page size
- Also implemented in a commercial ORDBMS
  - Used type extensibility to create array-like data types for sequences & user-defined functions for FSM
- Data Set: PIR
  - 250K protein tuples, 250MB
  - Segment Table: 10M tuples, 355MB
- Platform: 1.7GHz Xeon, Linux, 40GB SCSI disk

Complex Histogram Accuracy

- Query: \{<l 15 15> <? 0 X> <h 24 24>\}
- Experiment: Vary Gap Predicate

<table>
<thead>
<tr>
<th>Length of Gap Predicate (X)</th>
<th>Estimate</th>
<th>Actual</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>50</td>
<td></td>
<td></td>
</tr>
<tr>
<td>100</td>
<td></td>
<td></td>
</tr>
<tr>
<td>500</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4000</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Histogram Details
- Data Set: PIR
  - 100x100x50x3
  - Size: 5.8MB
  - Build time: 22 secs
  - Estimation time: 20 ms

Histogram accurate within 80% of the actual result
Query Performance

- Query: 9 Predicates
  \( P_1, G_1, P_2, G_2, P_3, G_3, P_4, G_4, P_5 \)
  \( P = \) predicate, \( G = \) Gap pred.
- Expt: Vary Selectivity of P’s
  Alternatives: S (0.3%), L (7%)

- Choice of algorithm is critical
- CSP sensitive to position of \( L \) pred.
- Choice of \( k \) in MISS critical
  Index Probe
  - reduce # protein tuples fetched
  - probe cost, sorting and merging
  Influenced by query and predicate selectivity

Query: 9 Predicates

More details in our VLDB’02 paper

Roadmap

- Background and Introduction
- Primary Structure Sequence Matching
- Querying on Secondary Structure
- PiQA: Integrated Query Algebra
- Summary
PiQA

- An algebra in PNF for queries on primary & secondary structures

- Examples:
  - Match the primary sequence “AAANBPPPPSDF”, but ignore mismatch in the segment “NBPPP” if it is on a loop
    \((P.p \ast "AAA") \lor ((P.p \ast "NBPPP") \lor (P.s \ast <L 5 5>)) \lor (P.p \ast "PSDF")\)
  - BLAST: Match n-grams + match extension + transitive closure
  - Find all occurrences of ‘TGCTGACTCAGCA’ within 4000 bps upstream of a ‘CA’ with ‘TATA’ 25-30 bps upstream of the ‘CA’
    \(M \ast \text{TATA} \lor \text{TGCTGACTCAGCA} \lor \text{TATA} \lor \text{CA}\)

PiQL?

Details in our SSDBM’03 paper
Roadmap

► Background and Introduction
► Primary Structure Sequence Matching
► Querying on Secondary Structure
► PiQA: Integrated Query Algebra
► Summary

Summary

► Bioinformatics applications (urgently) need declarative and efficient query processing tools
  - Current procedural methods reminiscent of pre-relational days
► Database researchers have a lot to contribute!
► Periscope is our effort in this direction
  - PiQA: algebraic framework for querying on primary and secondary structures
  - Primary Structure: OASIS
  - Declarative querying on secondary structure: Periscope/PS²
► Current Status
  - OASIS and Periscope/PS² currently (beta-)deployed at UM
  - Under development: “PiQA powered PiQL queries on Periscope”
► http://www.eecs.umich.edu/periscope